Software Design I
Season 2024-I
Workshop No. 4 — Behavioral Design Patterns & AntiPatterns

Eng. Carlos Andrés Sierra, M.Sc.
Computer Engineering
Universidad Distrital Francisco José de Caldas

Welcome to the fourth and final workshop of Software Design I course, congratulations for keep pushing.

As you remember, you had been hired as software engineer in a vehicle constructor company. You are checking the current version of the internal tool used to manage the vehicles created by the company, and you think *Class Diagram* (Figure 1) could be improved, also the code in the project repository made by the previous software engineer is not the best, you could do it better.

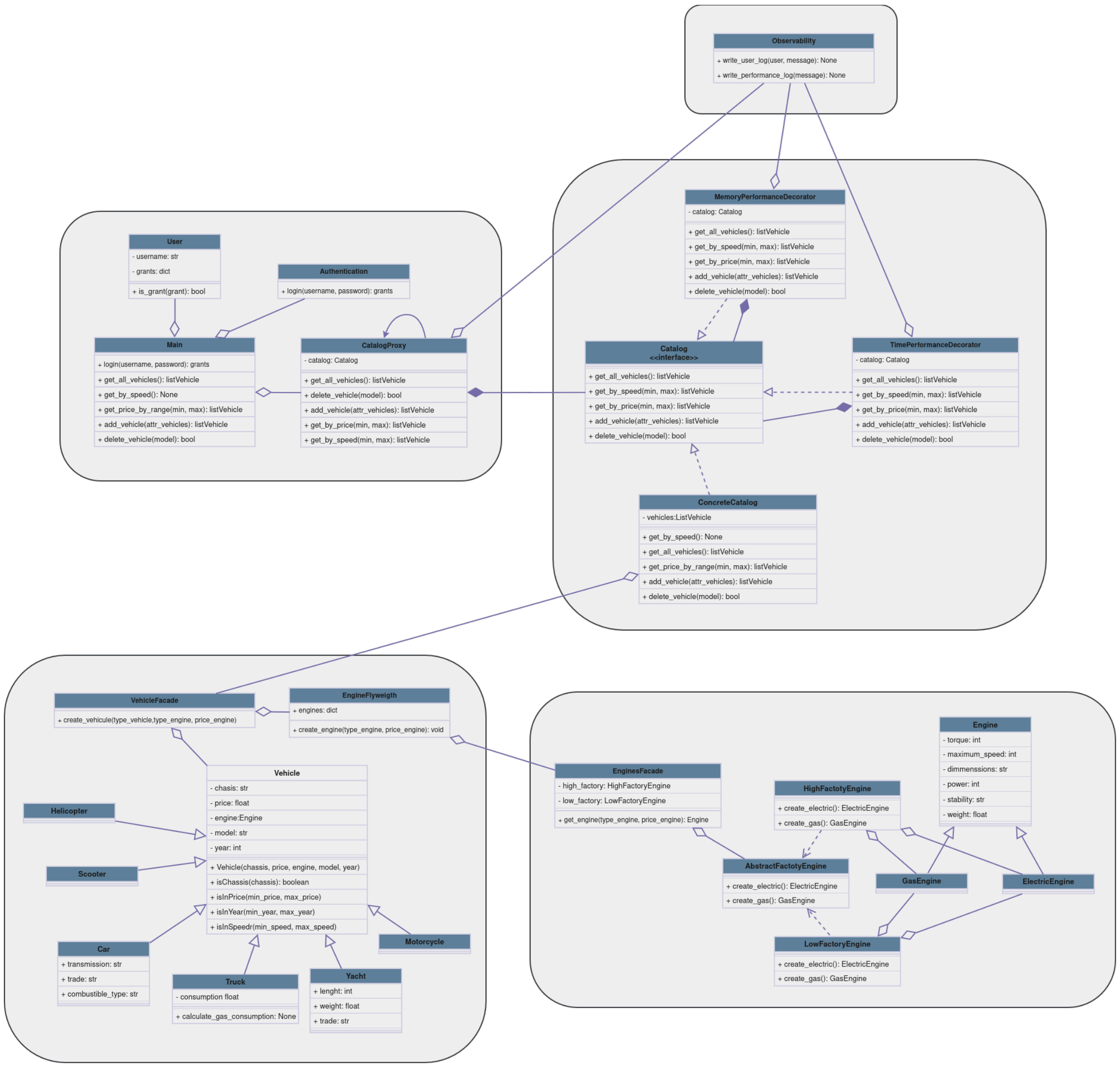
Thus, you must deliver a new version of this internal tool, improving anything you could. It means, improve documentation, code, designs, anythings.

Also, some requirements have been added after first final-user interactions. Changes are described as follows:

1. Application is not registering in a good way the actions of users. Check why is this failing and fix it.
2. There are just test cases for engines sub-system. You should add tests for all the other modules.
3. Check for code smells across the code, use *pylint* is a good idea. However, it is not enough, you must fix all the issues found after make a *code review.*
4. Code should be good formated and with the best comments you could write/update from the code.
5. Check the logs in order to figure out the functionalities with bad performance, and fix them.

Carlos Andrés Sierra, Computer Engineer, M.Sc. on Computer Engineering, Titular Professor at Universidad Distrital Francisco José de Caldas.

Any comment or concern related to this document could be send to Carlos A. Sierra at e-mail: *cavirguezs@udistrital.edu.co*


Observability
+ write_user_log(user, message): None
+ write_performance_log(message): None
User
- username: str
- grants: dict
+ is_grant(grant): bool
Authentication
+ login(username, password): grants
Main
+ login(username, password): grants
+ get_all_vehicles(): listVehicle
+ get_by_speed(): None
+ get_price_by_range(min, max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(model): bool
CatalogProxy
- catalog: Catalog
+ get_all_vehicles(): listVehicle
+ delete_vehicle(model): bool
+ add_vehicle(attr_vehicles): listVehicle
+ get_by_price(min, max): listVehicle
+ get_by_speed(min, max): listVehicle
MemoryPerformanceDecorator
- catalog: Catalog
+ get_all_vehicles(): listVehicle
+ get_by_speed(min, max): listVehicle
+ get_by_price(min, max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(model): bool
Catalog
<<interface>>
+ get_all_vehicles(): listVehicle
+ get_by_speed(min, max): listVehicle
+ get_by_price(min, max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(model): bool
TimePerformanceDecorator
- catalog: Catalog
+ get_all_vehicles(): listVehicle
+ get_by_speed(min, max): listVehicle
+ get_by_price(min, max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(model): bool
ConcreteCatalog
- vehicles:ListVehicle
+ get_by_speed(): None
+ get_all_vehicles(): listVehicle
+ get_price_by_range(min, max): listVehicle
+ add_vehicle(attr_vehicles): listVehicle
+ delete_vehicle(model): bool
VehicleFacade
+ create_vehicule(type_vehicle,type_engine, price_engine)
EngineFlyweigth
+ engines: dict
+ create_engine(type_engine, price_engine): void
Vehicle
- chasis: str
- price: float
- engine:Engine
- model: str
- year: int
+ Vehicle(chassis, price, engine, model, year)
+ isChassis(chassis): boolean
+ isInPrice(min_price, max_price)
+ isInYear(min_year, max_year)
+ isInSpeedr(min_speed, max_speed)
Helicopter
Scooter
Motorcycle
Car
+ transmission: str
+ trade: str
+ combustible_type: str
Truck
- consumption float
+ calculate_gas_consumption: None
Yacht
+ lenght: int
+ weight: float
+ trade: str
EnginesFacade
- high_factory: HighFactoryEngine
- low_factory: LowFactoryEngine
+ get_engine(type_engine, price_engine): Engine
HighFactotyEngine
+ create_electric(): ElectricEngine
+ create_gas(): GasEngine
Engine
- torque: int
- maximum_speed: int
- dimmenssions: str
- power: int
- stability: str
- weight: float
AbstractFactotyEngine
+ create_electric(): ElectricEngine
+ create_gas(): GasEngine
GasEngine
ElectricEngine
LowFactoryEngine
+ create_electric(): ElectricEngine
+ create_gas(): GasEngine
Figure 1. Current Class Diagram v3

6. Add a new feature to the application, now it should be able for users to be subscribed to a newsletter using an email. The newsletter should be sent by the admin anytime he/she wants, just with the information with the last five vehicles created (if there are less than five, just sent the complete catalog).

7. Cache memory is still do not implemented. It means, the last three searches made in the system shloud be saved into memory just to avoid make the same search again. Every time a vehicle is added, this memory should be shuffed.

8. Maybe an admin could make an error. So, provide an option to recovery the last vehicle deleted.

9. Add a new type of engine: a hybrid engine. This engine should have a new attribute:

`electric_power` (in *Watts*). The values for attributes of this new engine type will be defined by you but in the middle of the values of the other engines.

10. Implement a set of validations for the vehicles created. It means, a vehicle should have a valid `model name` (less than 30 characters, and just `numbers and letters`), a valid `year` (between 1990 and 2025), a valid `price` (between 20000000 COP and 500000000 COP), and a valid `chassis` (`A` or `B`).

You must deliver a technical report where a Class Diagram of your solution is provided, also all related updated additional documentation; here it is recommended to think in components, define a diagram por each component where connections with other components will be absolutely clear. Also, you must write about technical concerns and decisions you make to create the architecture you are proposing; this includes SOLID implementation analysis of your model, coupling, cohesion, and other software engineering principles. Finally, it is important you define why do you apply some patterns, and for any pattern you do not apply, you must explain why.

You must update the code and provide a simple *client interface* with a menu to create vehicles, show all vehicles registered, and make searches by maximum speed, type of combustion, and range of years, showing vehicles and them specific information independently of the type, login users, subscribe users, and send newsletters.